Fully Dynamic Maintenance of Euclidean Minimum Spanning Trees

David Eppstein

Department of Information and Computer Science
University of California, Irvine, CA 92717

Tech. Report 92-05

January 10, 1992

Abstract

We maintain the minimum spanning tree of a point set in the plane, subject to point insertions and deletions, in time $O(n^{5/6}\log^{1/2} n)$ per update operation. No nontrivial dynamic geometric minimum spanning tree algorithm was previously known. We reduce the problem to maintaining bichromatic closest pairs, which we also solve in the same time bounds. Our algorithm uses a novel construction, the *ordered nearest neighbors* of a sequence of points. Any point set or bichromatic point set can be ordered so that this graph is a simple path.

1 Introduction

Minimum spanning trees (abbreviated MSTs) form one of the most basic structures in computational geometry. They have applications in VLSI, network design, data clustering, heuristics for travelling salesman tours and minimum weight triangulations, and many other areas. The MST of a planar point set can be computed in time $O(n \log n)$ using Delaunay triangulations; algorithms for higher dimensional MSTs are also known [1].

Often we do not want to compute a single MST, but rather maintain the MST as the set of input points changes by the insertion of new points and the deletion of old points. No known online algorithm allowing both insertions and deletions (or even deletions only) was significantly better than the trivial $O(n \log n)$ method of recomputing the MST from scratch after each update. In this paper we provide the first such algorithm.

A natural approach would be to maintain one of the many linear-sized geometric graphs known to contain the MST. Such graphs include the Delaunay triangulation, the Gabriel graph, the relative neighborhood graph, and the graph of nearest neighbors in each of six directions. In the latter case, we could either consider pairs (x, y) in which y is the nearest neighbor to x, or *mutual* nearest neighbors, in which x is also the nearest neighbor to y. Unfortunately, each of these graphs can change by the insertion or deletion of $\Omega(n)$ edges when a single point is added or removed. Therefore none of these graphs can lead to a sublinear MST update algorithm.

Our approach is slightly different. Following Agarwal et al. [1], we reduce the problem to that of maintaining *bichromatic closest pairs*. This gives a graph of $\Theta(n \log^2 n)$ edges, of which $O(\log^2 n)$ edges can change in a single point update. Again, the graphs listed above (or the MST itself) can be used to find the bichromatic closest pair, but as above this does not lead to an efficient algorithm. We maintain the bichromatic closest pair using a new graph, the *ordered* nearest neighbor graph, in which we find the nearest point y to x among only those points y occuring after x in some sequence. For an appropriate sequence this graph is a simple path. If we perform k updates, the number of edge changes per update will be $O(k)$. We use some data structures to find these changes quickly, giving us our result.

1.1 New Results

In this paper we provide the first known fully dynamic geometric MST algorithms. We describe algorithms for the following versions of the problem.

- For any planar point set undergoing point insertions and deletions, the MST can be maintained in time $O(n^{5/6}\log^{1/2} n)$ per update.
- For any higher dimensional point set, the MST can be maintained in time $O(n \log n)$.
- For d-dimensional point sets in the rectilinear (L_1 or L_∞) metrics, the MST can be maintained in time $O(\sqrt{n}\log^{3d/2} n)$ per update.

Our algorithms use a data structure for maintaining graph spanning trees, together with a reduction to a problem of *bichromatic closest pairs*. For any two-colored planar point set undergoing insertions and deletions, we maintain the bichromatic closest pair in time $O(n^{5/6}\log^{1/2} n)$. We also use a novel construction, the *ordered nearest neighbors* of a sequence of points. Any point set or bichromatic point set can be ordered so that this graph is a simple path. We show that this ordering can be performed in time $O(n\log^{O(1)} n)$ time for uncolored point sets, and time $O(n^{3/2}\log^{1/2} n)$ for bichromatic point sets.

1.2 Related Work

Dynamic algorithms have been studied for many important geometric optimization problems, including closest pairs [6, 21, 22], diameter [6, 22], width [2], convex hulls [15, 18], smallest k-gons [10], and minimum spanning trees [8, 17].

For graph MSTs, the correponding problem (with edge insertions and deletions) can be solved efficiently for planar graphs [11, 12, 13], and for *offline* updates (in which the sequence of updates is known before any update is performed) [8, 9]. We use as a subroutine an algorithm for maintaining a graph MST under the more general model of online updates [12].

Less is known about the dynamic geometric MST problem. It is not hard to show that, if only insertions are allowed, the MST can be maintained in time $O(\log^2 n)$ per update. The same bound has recently been achieved for offline updates consisting of both insertions and deletions [8]; this can be improved for rectilinear metrics. Other work on geometric spanning trees has studied classification of the possible changes that can occur at a single update [4, 14, 17], computation of multiple spanning trees for a static point set [7], and problems of parametric point motion [5]. No known online algorithm allowing both insertions and deletions (or even deletions only)

was significantly better than the trivial $O(n \log n)$ method of recomputing the MST from scratch after each update.

2 Preliminaries

2.1 Bichromatic Closest Pairs

Our minimum spanning tree algorithm is based on the *bichromatic closest pair* problem. Suppose we are given a set of points, colored red and blue. The bichromatic closest pair is simply the two points x and y, with x red and y blue, minimizing the distance xy. We will need data structures that maintain the bichromatic closest pair as points are inserted or deleted. If we were performing insertions only, this would not be difficult. Vaidya [23] described a method for performing insertions in the red set, and both insertions and deletions in the blue set, in time $O(\sqrt{n} \log n)$ per operation. But no fully dynamic algorithm for this problem was previously known.

The bichromatic closest pair must be an edge of the MST (hence the planar bichromatic closest pair can be computed in $O(n \log n)$ time). Conversely, Agarwal et al. [1] showed that the MST can be computed in the same time (up to polylogarithmic factors) as the bichromatic closest pair. They used this to derive the best known algorithms for higher dimensional minimum spanning trees. We now state the planar version of their result:

Lemma 1 (Agarwal et al. [1]). *Given n points, we can form a collection of $O(n \log^2 n)$ bichromatic closest pair problems, so that each point is involved in $O(i)$ problems of size $O(n/2^i)$ $(1 \leq i \leq \log n)$ and so that each MST edge is the solution to one of the closest pair problems.*

Proof: If pq is an MST edge, and w is a double wedge having sufficiently small interior angle, with p in one half of w and q in the other, then pq must have the minimum distance over all such pairs defined by the points in w. Therefore if F is a family of double wedges with sufficiently small interior angles, such that for each pair of points (p, q) some double wedge $w(p, q)$ in F has p on one side and q on the other, then every MST edge pq is the bichromatic closest pair for wedge $w(p, q)$.

Suppose the interior angle required is $2\pi/k$. We can divide the space around each point p into k wedges, each having that interior angle. Suppose edge pq falls inside one particular wedge w. We find a collection of double wedges, with sides parallel to w, that is guaranteed to contain pq. By

repeating the construction k times, we are guaranteed to find a double wedge containing each possible edge.

For simplicity, assume that the sides of wedge w are horizontal and vertical. In the actual construction, w will have a smaller angle than $\pi/2$, but the details are similar. First choose a horizontal line with at most $n/2$ points above it, and at most $n/2$ points below. We continue recursively with each of these two subsets; therefore if the line does not cross pq, then pq is contained in a closest pair problem generated in one of the two recursive subproblems. At this point we have two sets, above and below the line. We next choose a vertical line, again dividing the point set in half. We continue recursively with the pairs of sets to the left of the line, and to the right of the line. If the line does not cross pq, then pq will be covered by a recursive subproblem. If both lines crossed pq, so that it was not covered by any recursive subproblem, then $w(p,q)$ can be taken to be one of two bichromatic closest pair problems formed by opposite pairs of the quadrants formed by the two lines.

The inner recursion (along the vertical lines) gives rise to one subproblem containing p at each level of the recursion, and each level halves the total number of points, so p ends up involved in one problem of each possible size $n/2^i$. The outer recursion generates an inner recursion at each possible size, giving i problems total of each size $n/2^i$. The construction must be repeated for each of the k wedge angles, multiplying the bounds by $O(1)$. □

2.2 Updating the MST in a Changing Graph

Since we reduce the geometric MST problem to a graph problem, we need an algorithm to update the MST in a changing graph. This need is satisfied by the following algorithm.

Lemma 2 (Frederickson [12]). *Given a graph subject to edge insertions and deletions, with m edges at any one time, the minimum spanning tree can be maintained in time $O(\sqrt{m})$ per update.* □

Lemma 3. *Suppose that we can solve the bichromatic closest pair problem in preprocessing time $P(n)$ and update time $T(n)$. Then we can maintain the minimum spanning tree of a planar point set in amortized time $O(\sqrt{n}\log^3 n + P(n)/n + T(n))$ per point insertion or deletion.*

Proof: By Lemma 1, each point update changes $O(\log^2 n)$ edges in a graph with $O(n\log^2 n)$ edges total. Therefore by Lemma 2, updating the MST in

this graph takes time $O(\sqrt{n}\log^3 n)$. The time to recompute the appropriate bichromatic closest pairs is $\sum iT(n/2^i)$, which is (if not dominated by the first term) simply $O(T(n))$. Finally, ¡Lemma 1 assumes that each recursion splits the point set exactly evenly. However it suffices if each recursive subproblem is at most a constant fraction of its parent size. This will remain true in the outermost level of the recursion for the first $O(n)$ updates, after which we must recompute the recursive structure to balance it again. The time to perform this recomputation, including recursively rebuilding smaller subproblems both when their parent is rebuilt and when the subproblem itself becomes unbalanced, can be expressed by a recurrence which either solves to $O(P(n)/n)$ or is again dominated by the first term. □

2.3 The Post Office Problem with Deletions

We use the *post office problem* as a subroutine at several points in our algorithm. In this problem, one is given a point set S, and a query point x; the problem is to find the nearest point to x in S. If S is unchanging, this can be done by computing the Voronoi diagram of S, in time $O(n\log n)$, and performing point location queries in the diagram in $O(\log n)$ time each.

In most of our uses of this problem, S will be undergoing update operations. We will only need to concern ourselves with point deletions, although insertions can be handled similarly. By a result of Aggarwal et al. [3], it takes $O(n)$ time to compute the changes to the Voronoi diagram resulting from the removal of a single point. After this computation, a point location data structure can be constructed in linear time [16]. Therefore we can take $O(n)$ time per update and $O(\log n)$ time per query to solve the post office problem with deletions. By a standard balancing technique, we can reduce the update time, at the expense of increasing the time per query.

Lemma 4. *For any $f(n) = O(n)$, we can solve the post office problem with deletions with preprocessing time $O(n\log f(n))$, time $O((n/f(n))\log f(n))$ per query, and time $O(f(n))$ per update.*

Proof: We form the points into $n/f(n)$ groups of $f(n)$ points each. For each group, we compute a Voronoi diagram and point location data structure. Each deletion forces the recomputation of these data structures for a single group, taking time $O(f(n))$. Each post office query performs a separate point location query in each group, in time $O(\log f(n))$ each. □

In our applications, $f(n)$ will always be of the form $n^\alpha \log^{1/2} n$, giving us a function of that form for the update time, and giving $O(n^{1-\alpha} \log^{1/2} n)$ as the query time. Initializing the data structure takes time $O(n \log n)$.

2.4 Amortized and Worst Case Times

Our algorithms work by computing some balanced structure (the divide and conquer tree in the reduction from minimum spanning trees to closest pairs, or the ordered nearest neighbor path in the closest pair algorithm), performing computations with the structure while it remains close to balanced, then recomputing the structure and starting afresh. Recomputations are costly in the time they take, but infrequent; therefore the average time per operation is small. Hence, we have algorithms with good *amortized* time complexity, but bad *worst-case* time complexity.

Our bounds can be tightened to worst case complexity, using the following trick (e.g. see [19]). Suppose we would normally reconstruct the data structure after every x updates. We instead keep two copies of the data structure, which are constructed at intervals of $x/2$ updates. One copy was created starting between $x/2$ and x updates prior to the present time, and is the one used for performing queries. The other copy is newer, and only partially constructed. At each update, we perform some more steps toward constructing the new copy. For the first $x/4$ updates, we create the data structure in the state it would have been first created in the original algorithm, splitting the work evenly among these updates. For the next $x/4$ updates, we catch up on the updates that took place while we were creating the data structure, by performing two updates at a time. After $x/2$ updates, the data structure is fully up to date, and we start using it to answer queries.

This works for our bichromatic closest pair algorithm, and gives us worst case bounds equal to the amortized bounds we claim. For the reduction from minimum spanning trees to closest pairs, the situation is more complicated. We have many different data structures, at different levels in the divide and conquer, which must each be recreated at different intervals as they become unbalanced. Performing the duplication described above at each of these levels, as well as at each level of each duplicate data structure, and so on, would increase the total time by more than a constant factor.

Instead, note that for subproblems involving fewer than $n^{2/3}$ points, we can recompute the bichromatic closest pair from scratch rather than maintaining our data structures, in total time $O(n^{2/3} \log^3 n)$ per update (since each update involves $O(\log^2 n)$ such problems), which is better than our

bound of $O(n^{5/6}\log^{1/2} n)$ for the larger subproblems. Second, we recompute the subproblems with between $n^{2/3}$ and $n^{8/9}$ points all at once, at intervals of $O(n^{2/3})$ updates. As the time to recompute our data structure will be $O(n^{3/2}\log^{1/2} n)$, the amortized recomputation time will be $O(n^{2/3}\log^{1/2} n)$. Finally, we recompute the remaining subproblems at intervals of $O(n^{8/9})$ updates, in average time $O(n^{11/18})$ per update.

These modifications do not change the amortized nature of our bounds, but they perform the recomputation at more manageable times. We now use the amortization elimination method above. We maintain two copies of the data structure, which switch roles at intervals of $O(n^{8/9})$ updates. At the level corresponding to subproblems of size $n^{8/9}$, we again duplicate the data structure, keeping two copies that switch roles after every $O(n^{2/3})$ of the updates that involve those subproblems. Each data structure is duplicated four times, so the total time will be proportional to the amortized bounds.

3 Rectilinear Metrics and Higher Dimensions

As an application of Lemma 3, we derive an immediate solution to minimum spanning tree maintenance for rectilinear metrics. Interdistance selection problems with this metric are noticably easier than with the Euclidean metric [20], and in fact the bichromatic closest pair can be maintained quite efficiently. This leads to the following result.

Theorem 1. *The L_1 (equivalently L_∞) metric MST can be maintained in time $O(\sqrt{n}\log^3 n)$ per point insertion or deletion.*

Proof: Lemma 1 works for this metric, again producing a collection of bichromatic closest pair problems formed by double wedges. We choose the wedge angles so that no vertical or horizontal line is interior to any double wedge. Then if (x, y) is the bichromatic closest pair, x and y must be closer to the centerpoint than the other points in the same wedge. So by using a priority queue for each wedge, the bichromatic closest pair can be maintained in time $O(\log n)$. The time bound claimed then follows from Lemma 3. □

This result generalizes to any higher dimension, for either the L_1 or L_∞ metrics. The time to maintain the bichromatic closest pair remains $O(\log n)$. The number of edges in the graph of closest pairs becomes $O(n\log^d n)$, and $O(\log^d n)$ of these edges change per update, giving a total time per update of $O(\sqrt{n}\log^{3d/2} n)$.

For an arbitrary higher-dimensional metric, we can maintain bichromatic closest pairs as follows. Simply maintain for each red point a priority queue of the distances to all blue points. Each update involves an insertion or deletion in $O(n)$ priority queues, and takes time $O(n \log n)$. The time to maintain the MST in the graph of closest pairs is smaller than this, so we have the following result:

Theorem 2. *In any dimension, the Euclidean MST can be maintained in time $O(n \log n)$ per point insertion or deletion.*

4 Ordered Nearest Neighbors

Suppose we are given a sequence of points $x_1, x_2, \ldots x_n$. This differs from the usual point set in that the points are ordered. We define the *ordered nearest neighbor* of a point x_i to be the nearest point to x in the set $\{x_j : j > i\}$. Note that if x_i and x_j are mutual nearest neighbors in the point set, and $i < j$, then x_j is the ordered nearest neighbor to x_i in the point sequence. The *ordered nearest neighbor graph* is a directed graph in which each point has an edge directed to its nearest neighbor. The ordered nearest neighbor graph can be computed in time $O(n \log^2 n)$ using standard decomposable search problem techniques, but we do not need such a result.

Similarly, given a sequence of points with two colors (red and blue) we define the *bichromatic ordered nearest neighbor* of a point x to be the nearest point y that appears after x in the sequence, and that has the opposite color to x. The bichromatic ordered nearest neighbor graph is defined in the obvious way. Again if x and y are mutual nearest neighbors, they will be connected by an edge in the ordered nearest neighbor graph. In particular the bichromatic closest pair can be found as such an edge.

Like the usual graph of nearest neighbors, ordered nearest neighbor graphs are in the form of a pseudo-forest; that is, a directed graph with out-degree at most one. However the in-degree of a point could be large. The following lemma tells us that, if we choose the sequence correctly, the in-degrees will also be small.

Lemma 5. *For any point set or bichromatic point set, the points can be ordered into a sequence, in time $O(n^{3/2} \log^{1/2} n)$, such that the ordered nearest neighbor graph forms a simple path.*

Proof: We choose x_1 arbitrarily. Assume inductively that we have chosen x_1 through x_i so that the nearest neighbor of each point x_j, among those

points later in the sequence or not yet chosen, is x_{j+1}. The graph then forms a path from x_1 to x_i. To extend this path by one more point, we simply choose x_{i+1} to be the nearest neighbor to x_i among the unchosen points.

We maintain a post office problem data structure for looking up each nearest neighbor among the unchosen points. When we choose a point we remove it from the structure. There are $n-1$ queries and a similar number of deletions; by Lemma 4 we can perform each operation in time $O(n^{1/2}\log^{1/2} n)$. Therefore the total time is $O(n^{3/2}\log^{1/2} n)$. □

5 Maintaining the Closest Pair

We can maintain the bichromatic closest pair of points by sequencing the points as above, and maintaining the ordered nearest neighbor graph of the point sequence. We insert all new points at the start of the sequence, so that they do not disrupt the connections among the remaining points. Call point x_i an *orphan* if either x_{i+1} was deleted, or if x_i was inserted since the last recomputation of the point sequence.

After k updates, there will be at most $2k$ orphans. The nearest neighbor to any non-orphan x_j will be x_{j+1}. Therefore the maximum in-degree in the nearest neighbor graph will be $2k+1$. In the worst case each orphan will change neighbors after each update, so $O(k)$ edges will change in the graph.

We recompute the point sequence, and create a data structure for the post office problem, after every $n^{2/3}$ updates, in $O(n^{5/6}\log^{1/2} n)$ amortized time per update. This lets us use the same amount of time to perform updates in the post office problem, so that we can perform the queries that allow us to recompute the nearest neighbor graph.

Lemma 6. *We can maintain the ordered nearest neighbor graph, starting with the sequence for which it is a path, and continuing through $O(n^{2/3})$ updates, in time $O(n^{5/6}\log^{1/2} n)$ per update.*

Proof: After each update we must recompute, for each orphan, the point that is its new nearest neighbor. Recall that the post office problem with deletions is solved by dividing the points into groups, and recomputing the Voronoi diagram of a single group after a point deletion. Here we maintain $n^{1/6}\log^{-1/2} n$ groups, so that a query takes total time $O(n^{1/6}\log^{1/2} n)$. We form the groups by dividing the path formed by the nearest neighbor graph into subpaths of the appropriate length. Then each orphan can find

its new ordered nearest neighbor by looking only in those groups corresponding to paths occurring after the orphan in the sequence. The orphan must also look in the group that it is itself contained in. To make this possible, we divide each group into two equal sized subgroups, and so on recursively through $O(\log n)$ levels of recursion. Each update involves one group at each level, and hence is dominated by the time for the outer level, which is $O(n^{5/6}\log^{1/2} n)$. Each orphan finds its new neighbor by looking in $O(n^{1/6}\log^{-1/2} n)$ whole groups, and $O(\log n)$ subgroups, so the time for all orphans to be updated is again $O(n^{5/6}\log^{1/2} n)$. □

Theorem 3. *We can maintain the bichromatic closest pair of a colored point set, and the minimum spanning tree of an uncolored point set, in amortized time $O(n^{5/6}\log^{1/2} n)$ per point insertion or deletion.*

Proof: The bichromatic closest pair must be an edge in the nearest neighbor graph, maintained as above. Maintaining a priority queue of the lengths of all edges takes an additional $O(n^{2/3}\log n)$ time per update. The minimum spanning tree bound then follows from Lemma 3. □

6 Conclusions

We have described algorithms for maintaining the minimum spanning tree of a changing point set, and the bichromatic closest pair of a colored point set, in time $O(n^{5/6}\log^{1/2} n)$ per operation. Neither of these problems was known to be solvable more efficiently than $O(n)$ per operation. This naturally raises the question of how far our algorithms can be extended or improved.

The post office problem with deletions is used at several points in our algorithm. The same problem with insertions instead of deletions is not difficult to solve in $O(\log^2 n)$ time per operation. A similar bound for deletions would greatly speed up our algorithms. However such an improvement seems difficult to attain.

Another subproblem that could be improved is computation of the order for which the nearest neighbor graph is a path. Our present algorithm takes time $O(n^{3/2}\log^{1/2} n)$, based on the post office problem with deletions. It would suffice to modify the graph's definition, retaining constant in-degree. For instance we could choose the nearest neighbor to x_i among those x_j with $j \geq i-1$, and compute the sequence by repeatedly choosing and removing the closest pair of points. The nearest neighbor to x_i would be either x_{i-1}

or x_{i+1}, and the in-degree would be at most two. For the non-bichromatic case (not used in our MST algorithm), the closest pair can be maintained, under point deletions, in polylogarithmic time [21, 22]. Therefore this sequence could be computed in time $O(n \log^{O(1)} n)$. However the best known algorithm for maintaining bichromatic closest pairs is the one in this paper, which would give a slower algorithm than the present method.

It is open whether the sequence we generate, or any other sequence for which the nearest neighbor in-degree is constant, can be found quickly in parallel. Our method of picking a point at a time and finding its nearest neighbor in the remaining set seems inherently sequential, but perhaps a more sophisticated approach will work.

One can generalize our approach to higher dimensions. In three dimensions we can find an ordering for which the ordered nearest neighbor graph is a path in time $O(n^{5/3} \log^{1/3} n)$. We could use this to compute bichromatic nearest neighbors in time $O(n^{16/15} \log^{O(1)} n)$, which at least improves the $O(n^{4/3} \log^{O(1)} n)$ bound for computing the MST from scratch [1]. However as we saw earlier, a different method solves this problem in time $O(n \log n)$. So further work is required before ordered nearest neighbors provide results in higher dimensions.

References

[1] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. 6th ACM Symp. Comput. Geom. (1990) 203–210.

[2] P.K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. 2nd ACM/SIAM Symp. Discrete Algorithms (1991) 449–458.

[3] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.* 4 (1989) 591–604.

[4] B. Aronov, M. Bern, and D. Eppstein. Arrangements of polyhedra and the 1-Steiner problem. In preparation.

[5] M.J. Atallah. Some dynamic computational geometry problems. *J. Comput. Appl. Math.* 11 (1985) 1117–1181.

[6] D. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. 30th IEEE Symp. Found. Computer Science (1990) 488–493.

[7] D. Eppstein. Finding the k smallest spanning trees. 2nd Scand. Worksh. Algorithm Theory, Springer-Verlag LNCS 447 (1990) 38–47.

[8] D. Eppstein. Offline algorithms for dynamic minimum spanning tree problems. 2nd Worksh. Algorithms and Data Structures, Springer-Verlag LNCS 519 (1991) 392–399.

[9] D. Eppstein. Persistence, offline algorithms, and space compaction. Tech. Rep. 91-54, Dept. Information and Computer Science, Univ. California, Irvine, 1991.

[10] D. Eppstein. Iterated nearest neighbors and finding minimal polygons. Manuscript, 1991.

[11] D. Eppstein, G.F. Italiano, R. Tamassia, R.E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. 1st ACM/SIAM Symp. Discrete Algorithms (1990) 1–11.

[12] G.N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.* 14 (1985) 781–798.

[13] H.N. Gabow and M. Stallman. Efficient algorithms for graphic matroid intersection and parity. 12th Int. Conf. Automata, Languages, and Programming, Springer-Verlag LNCS 194 (1985) 210–220.

[14] G. Georgakopoulos and C. Papadimitriou. The 1-Steiner tree problem. *J. Algorithms* 8 (1987) 122–130.

[15] J. Hershberger and S. Suri. Offline maintenance of planar configurations. 2nd ACM/SIAM Symp. Discrete Algorithms (1991) 32–41.

[16] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.* 12 (1983) 28–35.

[17] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. 7th ACM Symp. Comput. Geom. (1991) 239–249.

[18] M. Overmars and H. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.* 23 (1981) 166–204.

[19] M. Overmars and H. van Leeuwen. Dynamization of decomposible searching problems yielding good worst case bounds. 5th GI Fachtagung Theoretische Informatik, Springer-Verlag LNCS 104 (1981) 224–233.

[20] J. Salowe. L-infinity interdistance selection by parametric search. *Inf. Proc. Lett.* 30 (1989) 9–14.

[21] M. Smid. Maintaining the minimal distance of a point set in polylogarithmic time. 2nd ACM/SIAM Symp. Discrete Algorithms (1991) 1–6.

[22] K.J. Supowit. New techniques for some dynamic closest-point and farthest-point problems. 1st ACM/SIAM Symp. Discrete Algorithms (1990) 84–90.

[23] P.M. Vaidya. Geometry helps in matching. *SIAM J. Comput.* 18 (1989) 1201–1225.